Mode d’emploi de GCC

Christian Casteyde

Mode d’emploi de GCC

par Christian Casteyde

Historique des versions

Version 1.2.4 31/07/2001 Revu par : CC
Corrections orthographiques. Mise à jour pour GCC 3.0.3 et GDB 5.1.1.
Version 1.2.3 05/12/2000 Revu par : CC
Corrections orthographiques.
Version 1.2.2 01/10/2000 Revu par : CC
Corrections typographiques.
Version 1.2.1 11/09/2000 Revu par : CC
Passage au format de fichier SGML. Corrections mineures.
Version 1.2 01/06/2000 Revu par : CC
Corrections sur l’utilisation de l’analyseur de couverture et du profiler.
Version 1.1 25/05/2000 Revu par : CC
Précision sur les chiens de garde hardware dans les programmes multithreadés.
Version 1.0 23/05/2000 Revu par : CC
Version initiale.

Table des matières

Liste des tableaux

Préface

Ce document est un mode d'emploi simplifié des divers outils de développement réalisés par la Free Software Foundation dans le cadre du projet GNU. Le projet GNU est un projet visant à réaliser un système Unix complet et libre de droits d'utilisation. Libre signifie ici que l'on peut aussi bien utiliser que modifier le système, le distribuer librement de façon rémunérée ou non.

Les outils de développement GNU font partie des meilleurs outils Unix disponibles actuellement. Ils sont fournis en standard avec Linux, mais ils sont également disponibles pour de nombreuses plates-formes. Des portages ont également été réalisés pour DOS, Win32 et OS/2.

Ce document est distribué sous la licence FDL du projet GNU. C'est une licence libre, vous en trouverez un exemplaire en Annexe A. Vous pourrez trouver la dernière version de ce document sur mon site Web (http://casteyde.christian.free.fr).

Chapitre 1. Introduction

GCC est le nom générique de la suite d'outils de développement contenant, entre autres, le compilateur C/C++ GNU développé par la Free Software Foundation. C'est en fait un front-end qui permet d'utiliser les programmes de développement. Ce front-end est très souple et peut être utilisé pour les différentes phases de production des binaires : preprocessing des fichiers sources, compilation, assemblage et édition de liens. Il dispose d'un grand nombre d'options, et peut être paramétré à l'aide d'un fichier de configuration.

GCC existe en plusieurs versions. La dernière version officielle disponible est la 3.0.3. Cette version comprend un compilateur C/C++ et un compilateur Java. Le compilateur C++ est capable de comprendre la plupart des constructions de ce langage, excepté le mot-clé `export`. Il existe également une version plus ancienne mais nettement plus stable, la version 2.95.3, que l'on utilisera donc de préférence si l'on ne recherche pas les dernières fonctionnalités du langage et de la norme C++. Cette version ne comprend pas de compilateur Java.

La suite de développement GNU est certainement l'une des plus portables qui soit. Fondamentalement, elle a été conçue pour être utilisée sous UNIX. Cependant, il en existe également des versions pour d'autres systèmes d'exploitation. Ce document donne les notions relatives au développement avec le compilateur C/C++ de la suite de compilateurs GCC, pour les principales plates-formes pour lesquelles GCC a été porté. Ils donne également quelques informations spécifiques à chaque plate-forme, lorsque cela est nécessaire. Ces informations sont clairement identifiées dans le document.

SPÉCIFIQUE DOS : La version DOS est appelée DJGPP, elle correspond à la version 2.8.1. de GCC.

SPÉCIFIQUE WIN32 : Les versions Windows sont pour les plates-formes 32 bits de ce système. Il en existe trois version de GCC pour Windows.

La version fournie par Cygwin utilise une DLL d'émulation des appels UNIX sous l'API Win32 pour faire fonctionner la suite de développement GNU. Les programmes générés par Cygwin utilisent également cette DLL, ce qui fait de cette version l'outil idéal pour compiler des programmes UNIX sous Windows. Il est également possible de générer des programmes natifs pour Windows, qui n'utilisent pas la DLL d'émulation d'UNIX, pourvu que ces programmes n'utilisent pas la librairie C++.

Les deux autres versions utilisent le portage de la librairie C mingw32. Ces deux versions sont liées aux deux librairies C standard du système `CRTDLL.DLL` ou `MSVCRT.DLL`. Les programmes créés sont donc des programmes natifs (que ce soient des programmes C ou C++). Ces deux versions utilisent le portage réalisé par Cygwin (en particulier les extensions du linker pour gérer la plate-forme Win32) et ont été recompilées avec mingw32.

Comme toutes ces versions sont réalisées à partir des mêmes fichiers sources, toutes les options de ces plates-formes sont identiques.

Chapitre 2. Droits d’utilisation

Tous les programmes de la suite de développement sont soumis à la GPL (General Public License). La librairie GNU C++ est soumise à la LGPL (Library General Public License).

> **Note :** La GPL implique un certain nombre de droits et d’obligations pour leurs utilisateurs. Le but principal de cette licence est de garantir le droit à échanger, utiliser et modifier librement les programmes qui sont soumis à cette licence. Ceci signifie que les distributeurs de ces programmes doivent également fournir les sources, ou au moins suffisamment d’information pour que les récipiendaires puissent obtenir les sources librement. Bien entendu, toute personne désirant modifier le logiciel doit se plier aux exigences de la GPL, c’est à dire que la modification doit être signalée, que la version modifiée doit elle-même être soumise à la GPL, et que la version originale doit être accessible aux récipiendaires. Il en va de même pour les logiciels qui utilisent une partie du code licencié sous les termes de la GPL. Il va donc de soi que le fait d’être libre pour un logiciel supprime toute garantie quant à son utilisation. En effet, le code source peut avoir été modifié, et un dysfonctionnement de sa part ne peut être imputé à l’auteur du logiciel.
>
> La LGPL s’applique spécifiquement aux bibliothèques logicielles. Elle permet de faire la distinction entre l’utilisation d’une bibliothèque, qui est à la convenance de l’utilisateur, et l’utilisation d’une partie du code source d’un logiciel soumis à la GPL. Elle permet à des développeurs de programmes non libres d’utiliser des bibliothèques soumises à la LGPL.
>
> La LGPL stipule que tout programme qui utilise la bibliothèque n’est pas soumise à la GPL, puisqu’il ne contient pas de code appartenant à cette bibliothèque. Cependant, la version compilée de ce programme contient une partie du code de la bibliothèque, et est donc soumis à la LGPL. La version compilée peut être distribuée selon les termes de son auteur, mais elle doit impérativement être fournie avec le code source de la bibliothèque, y compris les modifications qui y ont été apportées, et avec le code source ou le code objet du programme qui utilise la bibliothèque. Cette obligation a pour but d’assurer la liberté de tout un chacun de modifier la bibliothèque et d’effectuer l’édition de lien et du logiciel avec cette version modifiée et son déboguage. Le distributeur n’est toutefois pas tenu d’assurer une édition de lien correcte si les fichiers d’en-tête de la bibliothèque ont été modifiés.
>
> Ces licences impliquent que vous pouvez utiliser les compilateurs GCC pour développer des produits libres ou commerciaux. Cependant, si vous utilisez une bibliothèque soumise à la LGPL, vous devez fournir au moins les fichiers objets de votre produit en plus des sources de la bibliothèque utilisée, afin que les utilisateurs puissent toujours avoir le droit de modifier la bibliothèque et d’utiliser cette version modifiée avec votre produit. Vous pouvez également lier vos exécutables avec les bibliothèques chargeables dynamiquement afin de séparer votre code de celui des bibliothèques utilisées.
>
> La bibliothèque GNU libgpp est soumise à la LGPL (classes d’entrées/sorties C++). Cette bibliothèque est en cours de remplacement par la librairie libstdc++ (ou libstdcxx), qui est également soumise à la LGPL (ce remplacement a été réalisé avec les versions 3.0 et ultérieures de GCC). Cependant, les exécutables liés avec cette librairie ne sont pas soumis à la LGPL à titre exceptionnel.
>
> Les compilateurs GNU sont fournis également avec la librairie standard C++ (STL). Celle-ci est copyrightée par Hewlett Packard. Elle peut être librement utilisée, sans aucune contrainte. Elle n’est fournie sans aucune garantie. De plus, les librairies C fournies avec les versions portées de GCC sont copyrightées par les auteurs respectifs du portage. Leur utilisation est soumise à la notice de copyright fournie par ces auteurs.
>
> DJGPP requiert que l’utilisateur puisse trouver DJGPP librement et que votre programme mentionne dans sa documentation et sa publicité la notice suivante :

```
This product includes software developed by the University of
California, Berkeley and its contributors.
```

En effet, une partie de la librairie C de DJGPP est utilise du code copyrighté par BSD. Mingw32 est librement utilisable (c'est une bibliothèque freeware). Cygwin32 est également librement utilisable.

Chapitre 3. Principaux outils

Les outils disponibles sont les suivants :

Tableau 3-1. Outils de développement GNU

Outil	Description
gcc	Front-end pour le compilateur GNU C.
g++	Front-end pour le compilateur GNU C++.
c++	Alias de g++.
cpp	Préprocesseur C.
cc1	Compilateur C.
cc1plus	Compilateur C++.
gasp	Préprocesseur pour l'assembleur.
as	Assembleur.
ar	Archiveur.
ld	Éditeur de liens.
make	Programme de gestion des dépendances pour la construction des projets.
gdb	Débogueur symbolique.
strip	Extracteur d'informations symboliques de déboguage.
gperf	Profiler.
gcov	Analyseur de taux de couverture.
g77	Traducteur de Fortran 77 en C.
dlltool	Outil de génération des tables d'exportation et des bibliothèques d'importation des DLLs pour Windows.
windres	Compilateur de fichiers de ressources pour Windows.

gcc est un front-end qui permet d'appeler le préprocesseur, le compilateur, l'assembleur et le linker pour les langages C et C++. Les opérations que ce programme réalise sont les suivantes :

- appel du préprocesseur C (nommé **cpp**) ;
- appel du compilateur C (nommé **cc1**), ou C++ (**cc1plus**) selon le type de fichier à compiler (les fichiers .c sont compilés en C, les fichiers .C, .cc et .cpp sont compilés en C++). Les compilateurs génèrent un fichier assembleur ;
- appel de l'assembleur (**as**) pour générer le fichier objet, si l'option `-S` n'est pas spécifiée ;
- appel de l'éditeur de liens (**ld**) pour générer l'exécutable ou la bibliothèque, si l'option `-c` n'est pas spécifiée. Dans ce cas, la librairie C standard est incluse par défaut dans la ligne de commande de l'édition de liens. La librairie C++ n'est en revanche pas incluse.

g++ est un front-end pour le langage C++, qui effectue le même travaille que gcc, mais qui ajoute également les librairies C++ à la ligne de commande de l'éditeur de liens. **c++**, s'il est présent, est le même programme que **g++**, fourni sous un autre nom.

SPÉCIFIQUE WIN32 : L'éditeur de liens n'est pas capable de générer des bibliothèques dynamiques pour Windows (DLL). En effet, les DLL de Windows contiennent ce que l'on appelle une table de fonctions exportées, qui n'est rien d'autre qu'un tableau de pointeurs vers les fonctions exportées de la DLL. Cette table est utilisée par le chargeur de Windows pour résoudre dynamiquement les liens vers les fonctions de la DLL dans les programmes qui l'utilisent. Comme **ld** n'est qu'un éditeur de liens, il ne génère pas ces tables, et il faut recourrir à l'outil **dlltool**. Celui-ci peut créer un fichier objet contenant la table des fonctions exportées, que l'on utilisera ensuite pour effectuer l'édition de liens complète. L'outil **dlltool** permet également de générer les bibliothèques d'importation statiques pour les DLL existantes, afin de pouvoir les charger automatiquement au chargement d'un programme.

Les chemins passés aux programmes appelés peuvent être spécifiés aussi bien au format DOS qu'au format UNIX. Cependant, il est préférable d'utiliser le format UNIX, car dans les fichiers makefile, le séparateur \ ne peut être obtenu qu'en le doublant en raison de sa signification particulière de fin de ligne. Ce problème n'apparaît pas avec les chemins UNIX, qui utilisent le séparateur /.

SPÉCIFIQUE DOS : Même remarque pour les chemins que pour Windows. De plus, les noms de fichiers doivent être limités à 8 lettres et d'une extension au plus, de 3 lettres maximum.

La suite de développement GNU utilise un fichier spécial pour permettre à l'utilisateur de définir de nouvelles options pour ses programmes. Ce fichier est souvent placé dans le répertoire où se trouvent les bibliothèques, il se nomme `specs`. Il permet, pour chaque programme, de définir les nouvelles options, les conditions d'applications et ce par quoi ces options doivent être remplacées dans la ligne de commande de l'outil.

L'utilisation des différents outils est décrite dans la suite de ce fichier.

Chapitre 4. Utilisation de GCC

gcc dispose des options classiques de la plupart des front-end de compilateurs. Les principales options sont définies ci-dessous :

Tableau 4-1. Options de gcc

Option	Signification
`--help`	Affiche l'aide de GCC.
`--version`	Donne la version de GCC.
`-E`	Appelle le préprocesseur. N'effectue pas la compilation.
`-S`	Appelle le préprocesseur et effectue la compilation. N'effectue pas l'assemblage ni l'édition de lien. Seuls les fichiers assembleur (« .S ») sont générés.
`-c`	Appelle le préprocesseur, effectue la compilation et l'assemblage, mais ne fait pas l'édition de lien. Seuls les fichiers objets (« .o ») sont générés.
`-o nom`	Fixe le nom du fichier objet généré lors de la compilation d'un fichier source.
`-g`	Génère les informations symboliques de déboguage.
`-fexceptions`	Active la gestion des exceptions C++.
`-fpic`	Génère du code relogeable. Cette option est nécessaire pour la compilation des fichiers utilisés dans une DLL ou un fichier chargeable dynamiquement.
`-On`	Indique le niveau d'optimisation (`n` peut prendre les valeurs allant de 0 à 3, ou « `s` » pour optimiser la taille des binaires).
`-mcpu=cpu`	Indique le type de processeur pour lequel le code doit être optimisé. Le code fonctionnera sur tous les processeurs de la famille de ce processeur.
`-march=cpu`	Indique le type de processeur pour lequel le code doit être généré. Le code généré sera spécifique à ce processeur, et ne fonctionnera peut-être pas sur un autre modèle de la même famille. Cette option active automatiquement l'option `-mcpu` avec le même processeur.
`-pipe`	Utilise les pipes systèmes au lieu des fichiers temporaires pour les communications entre le préprocesseur, le compilateur et l'assembleur.
`-w`	Supprime tous les warnings.
`-W`	Active les warnings supplémentaires.
`-Wall`	Active tous les warnings possibles.
`-mwindows`	Crée un exécutable GUI Windows.
`-mdll`	Crée une DLL Windows.
`-fvtable-thunks`	Utilise le mécanisme des tables de fonctions virtuelles. Cette option est nécessaire pour utiliser les interfaces COM sous Windows.

La plupart des autres options ne sont pas gérées directement par **gcc**. Elles sont directement communiquées aux programmes appelés par lui. Les options de ces programmes peuvent donc souvent être utilisées avec **gcc**, voir la documentation de ces programmes pour plus de renseignements.

> **SPÉCIFIQUE WIN32 :** Les options -mwindows et -dll ne sont pas des options natives de la suite de développement GNU. Elles sont définies dans le fichiers specs de la version Windows.

Il est également possible de passer des options spécifiques aux programmes appelés par **gcc** à l'aide de l'option -W<lettre>. Les options les plus utiles sont les suivantes :

Option	Signification
-Wa,<options>	Passe les options suivantes à l'assembleur. Les options doivent être séparées par des virgules.
-Wp,<options>	Passe les options suivantes au préprocesseur. Les options doivent être séparées par des virgules.
-Wl,<options>	Passe les options suivantes à l'éditeur de liens. Les options doivent être séparées par des virgules.

gcc prend directement en ligne de commande les fichiers à traiter. L'ordre des fichiers n'a pas d'importance, sauf pour les fichiers de bibliothèques. Les bibliothèques doivent être passées à la fin, parce qu'elles ne sont utilisées que si l'éditeur de liens y trouve des symboles non résolus au moment de l'utilisation de la bibliothèque.

Chapitre 5. Le préprocesseur

Le préprocesseur est le premier programme appelé par gcc. Son rôle est de supprimer les commentaires, d'inclure les fichiers spécifiés avec la directive « `#include` » et d'évaluer les macros. À l'issue de l'exécution du préprocesseur, le fichier peut être compilé directement sans autre traitement par le compilateur.

La ligne de commande du préprocesseur est donnée ci-dessous :

```
cpp options source destination
```

où `options` représente les diverses options que l'on peut passer au préprocesseur, `source` est le nom du fichier source à traiter et `destination` est le nom du fichier de sortie.

Les principales options du préprocesseur sont les suivantes :

Tableau 5-1. Option du préprocesseur

Option	Signification
`--help`	Affiche l'écran d'aide du préprocesseur.
`-I répertoire`	Ajoute le répertoire `répertoire` à la liste des répertoires. Ce répertoire est ajouté en tête de liste.
`-D nom`	Déclare l'identificateur `nom`.
`-D macro=définition`	Définit la macro `macro`.
`-U nom`	Supprime la définition du symbole `nom`.
`-trigraphs`	Autorise l'utilisation des trigraphes C.

Chapitre 6. Le compilateur

Le compilateur appelé par **gcc** est soit **cc1**, si le fichier source est un fichier C, soit **cc1plus**, s'il s'agit d'un fichier C++. Seules les principales options sont décrites ici. Pour plus de renseignements, consulter la documentation de GCC, ou demandez l'écran d'aide du compilateur.

Tableau 6-1. Options du compilateur C

Option	Signification
`--help`	Affiche l'écran d'aide du compilateur.
`--version`	Affiche la version du compilateur.
`-o nom`	Donne le nom du fichier de sortie.
`-g`	Génère les informations symboliques de déboguage.
`-On`	Active les optimisations (`n` allant de `0` à `3`, ou « `s` » pour optimiser la taille du code généré).
`-f<option>`	Active une option.
`-ansi`	Demande la compilation du code ANSI.
`-pedantic`	Affiche les warnings requis par la norme ANSI du langage.
`-pedantic-errors`	Génère les erreurs requis par la norme ANSI du langage.
`-w`	Supprime tous les warnings.
`-W`	Active les warnings supplémentaires.
`-Wall`	Active tous les warnings possibles.
`-p`	Active le profiling des fonctions.
`-a`	Active le profiling des blocks.
`-ax`	Active le profiling des branchements.
`-fPIC`	Génère du code indépendant de son adresse de chargement (nécessaire pour les bibliothèques dynamiques, dont le chargement peut être réalisé par le système n'importe où en mémoire a priori).

Le compilateur prend en ligne de commande le fichier à compiler. Il génère le code assembleur résultant de la compilation.

Chapitre 7. L’assembleur

L’assembleur GNU a été créé pour permettre l’assemblage des fichiers assembleurs générés par le compilateur. Ceci signifie que toutes les constructions des autres assembleurs conçus spécialement pour utiliser l’assembleur en tant que code source d’un programme ne sont pas forcément gérées par l’assembleur GNU. En revanche, la syntaxe de l’assembleur reste à peu près identique entre les différentes versions pour les différentes plates-formes. Bien entendu, les mnémoniques sont toujours spécifiques au processeur.

L’assembleur GNU ne sera donc utilisé qu’avec le compilateur GNU pour la plate-forme correspondante en pratique. Si l’on veut réaliser du code assembleur soi-même, il est recommandé d’utiliser un assembleur spécialisé.

Les principales options de l’assembleur sont les suivantes :

Tableau 7-1. Options de l’assembleur

Option	Signification
`--help`	Affiche l’écran d’aide de l’assembleur.
`--version`	Affiche le numéro de version de l’assembleur.
`--statistics`	Affiche les statistiques sur l’exécution de l’assembleur.
`-I répertoire`	Ajoute un répertoire pour la recherche des fichiers inclus.
`-o nom`	Donne le nom du fichier objet à générer.
`-W`	Supprime l’affichage des warnings.

En général, l’assemblage d’un fichier se fera avec la ligne de commande suivante :

```
as -o objet source
```

où `objet` est le nom du fichier objet résultat et `source` le nom du fichier assembleur à assembler.

Chapitre 8. L’archiveur

L’archiveur permet de regrouper plusieurs fichiers objets dans un seul fichier bibliothèque. Les principales commandes de **ar** sont l’ajout, la suppression et l’extraction des fichiers objets de la bibliothèque. Toutes les commandes prennent en paramètre une lettre indiquant l’action à réaliser, avec éventuellement quelques modificateurs, et le fichier bibliothèque sur lequel **ar** doit travailler. Les noms des fichiers objets doivent parfois être fournis également.

Pour créer une bibliothèque ou lui ajouter des fichiers, on utilisera la ligne de commande suivante :

```
ar -r bibliothèque fichiers
```

où `bibliothèque` est le nom de la bibliothèque et `fichiers` est la liste des fichiers objets à ajouter. **ar** n’ajoute pas les extensions aux noms des fichiers, le nom de la bibliothèque doit donc comprendre l’extension « .a » et le nom des fichiers objets l’extension « .o ». Si l’archive contient déjà les fichiers objets passés en paramètre, ceux-ci sont remplacés. Si l’on ne veut remplacer que les fichiers de l’archive dont la date et l’heure de création est antérieure à la date et l’heure de création des fichiers à ajouter, il faut utiliser l’option `-ru` à la place de l’option `-r`.

Pour supprimer un fichier objet d’une bibliothèque, il faut utiliser la ligne de commande suivante :

```
ar -d bibliothèque fichier
```

Pour lister les fichiers objets d’une archive, il faut utiliser la commande suivante :

```
ar -t bibliothèque
```

Pour extraire des fichiers objets d’une archive, il faut utiliser la ligne de commande suivante :

```
ar -x bibliothèque fichier
```

Si aucun fichier n’est spécifié, tous les fichiers sont extraits de l’archive. Si l’on fixer la date et l’heure de création des fichiers extraits à leur date d’archivage, il faut utiliser l’option `-xo` à la place de `-x`.

Chapitre 9. L'éditeur de liens

L'éditeur de liens permet de regrouper les fichiers .o et les bibliothèques .a pour former un exécutable ou une nouvelle bibliothèque. La plupart des options spécifiques au système cible sont indiquées lors de la phase d'édition de liens.

La syntaxe de l'éditeur de liens est relativement simple. Il prend en ligne de commande la liste des fichiers à utiliser pour la résolution des symboles, dans l'ordre dans lequel ils doivent être utilisés. Les principales options sont décrites ci-dessous :

Tableau 9-1. Options de l'éditeur de liens

Option	Signification
`--help`	Affiche l'écran d'aide de l'éditeur de liens.
`--version`	Donne le numéro de version de l'éditeur de liens.
`-o nom`	Spécifie le nom du fichier généré par l'éditeur de liens.
`-L chemin`	Ajoute un répertoire à la liste des répertoires utilisés pour la recherche des bibliothèques.
`-l bibliothèque`	Ajoute une bibliothèque dans la liste des fichiers à utiliser pour l'édition de liens.
`-e symbole`	Spécifie le symbole devant être utilisé comme point d'entrée dans le fichier généré.
`-i`	Utilise l'édition de lien incrémentale.
`-s`	Supprime tous les symboles du fichier généré.
`-S`	Supprime les informations symboliques de déboguage du fichier généré.
`--sybsystem système`	Permet d'indiquer le sous-système pour lequel l'exécutable est généré.
`--rpath chemin`	Permet de donner le chemin dans lequel l'éditeur de liens doit rechercher les bibliothèques dynamiques dont dépend la bibliothèque ou le programme dont on réalise l'édition de liens. Cette option permet de réaliser une édition de lien avec des symboles externes se trouvant dans des bibliothèques chargeables dynamiquement autre que celles installées dans le système.
`--soname nom`	Permet de donner un nom interne à la librairie autre que le nom du fichier généré par l'éditeur de lien. Ce nom est utilisé lors de l'édition de liens dynamiques afin de déterminer quelle bibliothèque doit être utilisée avec le programme en cours de chargement.
`--dll`	Permet de créer une DLL Windows.
`--base-file fichier`	Permet de créer un fichier map des symboles des DLL de Windows.

Note : On notera que bien que les fichiers contenant les bibliothèques ont tous la forme `libNNN.a` ou `libNNN.so`, seul le nom `NNN` de la bibliothèque doit être utilisé dans la ligne de commande du linker. Par exemple, pour utiliser la bibliothèque `libc.a`, il faudra utiliser l'option `-lc`.

SPÉCIFIQUE WIN32 : Les bibliothèques chargeables dynamiquement ne se nomme pas de la même manière sous Windows que sous Unix. En effet, l'extension de ces bibliothèques est `.DLL` d'une part, et les fichiers de bibliothèques ne disposent pas du suffixe `lib`. De plus, l'édition de liens avec une bibliothèque dynamique ne se fait pas directement, mais plutôt avec une librairie d'importation statique.

Les options `-mdll` et `-mwindows` de **gcc** activent respectivement les options `--subsystem windows` et `--dll` de l'éditeur de liens. La première option effectue l'édition de liens avec les bibliothèques `user32`, `gdi32` et `comdlg32`, et spécifie le fichier `crt1.o` comme fichier de démarrage. Elle permet donc de réaliser un exécutable GUI Windows. La deuxième option spécifie comme point d'entrée la fonction `DllMainCRTStartup`, et le comme fichier de démarrage le fichier `dllcrt1.0`. Cette option permet donc de réaliser une DLL Windows. Aucune option particulière ne doit être utilisée pour réaliser un programme Win32 en mode console.

L'option suivante `--base-file fichier` est disponible afin de permettre la création de DLL relogeable. Elle permet de demander à l'éditeur de liens de générer un fichier contenant les informations sur les adresses relatives des symboles du fichier objet passé en paramètre par rapport à leur adresse de base. Ce fichier peut être exploité par l'outil **dlltool**, afin de générer le fichier contenant la table d'exportation des symboles de la DLL. Cette table doit être communiquée en ligne de commande, comme un fichier objet classique, lors de l'édition de lien finale de la DLL. L'outil **dlltool** sera décrit plus en détail plus loin.

Enfin, le fichier `specs` fourni avec les versions Win32 de GCC ajoute les bibliothèques suivantes à la ligne de commande de l'éditeur de liens :

- `kernel32`
- `advapi32`
- `shell32`

Ces bibliothèques sont les bibliothèques de base pour tous les programmes Windows.

Chapitre 10. Utilisation de make

make permet d'interpréter un fichier de dépendance entre les différents éléments nécessaires à la création d'un projet. Ce fichier est appelé « `makefile` ». En fait, il contient les relations entre fichiers qui permettent d'établir les dépendances nécessaires à la création d'autres fichiers. Si, pour créer un fichier `A`, il faut utiliser un fichier `B`, alors il va d'abord falloir créer le fichier `B`.

Le travail de **make** est essentiellement de vérifier les dates des fichiers pour déterminer quels sont ceux qui sont à jour et ceux qu'il faut recréer afin de générer le projet. Pour chacun des fichiers à recréer, une ligne de commande est exécutée.

La syntaxe des fichiers makefile dépasse le cadre de ce document. En revanche, l'utilisation de **make** lui-même est élémentaire, puisqu'il suffit de l'appeler en passant en paramètre le nom du projet à créer :

```
make cible
```

En général, les fichiers makefile contiennent une cible nommée « `all` », qui permet de générer tout le projet.

En réalité, **make** peut être utilisé dans un autre contexte que la programmation : il permet d'effectuer des commandes de mises à jour de fichiers qui dépendent de la mise à jour d'autres fichiers.

Chapitre 11. strip

strip est un petit utilitaire qui permet de retirer les informations symboliques de déboguage d'un fichier. Son utilisation est des plus simples :

```
strip fichier
```

Le fichier est modifié à l'issue de cette commande. Il ne contient plus d'informations symboliques de déboguage. Rappelons que ces informations sont générées par le compilateur si l'option `-g` est passée en ligne de commande. On notera toutefois que le fait de ne pas mettre cette option n'implique pas que le résultat de l'édition de lien ne contient pas de telles informations, en effet, les bibliothèques utilisées par l'éditeur de liens peuvent avoir été compilées avec cette option.

Les principales options de **strip** sont les suivantes :

Tableau 11-1. Options de strip

Option	Signification
`--help`	Affiche l'écran d'aide de strip.
`--version`	Affiche le numéro de version de strip.
`--strip-all`	Supprime toutes les informations de symboliques.
`--strip-debug`	Supprime les infomations symboliques de déboguage.
`--strip-unneeded`	Supprime toutes les informations symboliques, sauf celles nécessaires au relogement des bibliothèques.

Chapitre 12. Windres

L'outil **windres** permet de compiler les fichiers ressources « .RC » de windows en fichiers objets, que l'éditeur de lien peut utiliser pour créer des exécutables ou des bibliothèques dynamiques. Les principales options que ce programme acceptent sont décrites ci-dessous :

Tableau 12-1. Options de windres

Option	Signification
`--help`	Affiche l'écran d'aide de windres.
`--version`	Donne le numéro de version de windres.
`-i source` ou `--input-file source`	Nom du fichier .RC de ressource à compiler.
`-o objet` ou `--output-file objet`	Nom du fichier objet à produire.
`--include-dir répertoire`	Ajoute le répertoire `répertoire` à la liste des répertoires pour les fichiers d'inclusion.
`--define symbole`	Définit le symbole `symbole` pour la compilation conditionnelle.

windres définit automatiquement l'identificateur `RC_INVOKED`, qui peut être utilisé dans la compilation conditionnelle des fichiers d'en-tête.

L'éditeur de liens n'accepte qu'un seul fichier objet pour les fichiers de ressources. Il faut donc placer toutes les ressources dans le même fichier de ressources.

Chapitre 13. dlltool

dlltool est un outil utilisé pour la création de DLL windows et la création de bibliothèques d'importation pour les DLL existantes. La plupart de ses options sont disponibles sous deux formes, une forme courte et une forme détaillée. Ces deux formes sont strictement équivalentes. Les options les plus utiles sont les suivantes :

Tableau 13-1. Options de dlltool

Option	Signification
`--help`	Affiche l'écran d'aide de dlltool.
`-V` ou `--version`	Affiche le numéro de version de dlltool.
`-e export` ou `--output-exp export`	Permet de générer un fichier `export` contenant la table des symboles exportés par la DLL. Ce fichier est utilisé par l'éditeur de liens lors de la création des DLL.
`-l bibliothèque` ou `--output-lib bibliothèque`	Permet de générer une bibliothèque d'importation `bibliothèque` pour permettre l'utilisation d'une bibliothèque dynamique.
`-D nom` ou `--dll-name nom`	Permet de donner le nom de la DLL pour les références à cette DLL dans les chemins stockés dans les fichiers générés.
`-d def` ou `--input-def def`	Permet de donner le nom du fichier .DEF à utiliser.
`-z def` ou `--output-def def`	Permet de donner le nom du fichier .DEF à générer.
`--export-all-symbols`	Permet d'exporter tous les symboles d'un fichier objet dans le fichier `.DEF` généré.
`-b base` ou `--base-file base`	Permet de donner le nom du fichier d'informations à utiliser pour le relogement des DLL relogeables.

La première utilisation de **dlltool** est de générer les bibliothèques d'importation pour les DLL existantes. Pour cela, il faut disposer du fichier `.DEF` contenant la déclaration des symboles exportés par la DLL. Si l'on a développé cette DLL, ce fichier `.DEF` est disponible, sinon, il faut le générer. Pour cela, un petit utilitaire nommé **impdef** peut être utilisé. Sa syntaxe est très simple :

```
impdef dll
```

où `dll` est le nom de la DLL à lire. Le fichier `.DEF` est affiché à l'écran, il suffit de rediriger la sortie de **impdef** pour obtenir ce fichier sur disque.

Une fois que le fichier `.DEF` créé, il suffit d'appeler **dlltool** avec la ligne de commande suivante :

```
dlltool --dllname dll --input-def def --output-lib bibliothèque
```

où `dll` est le nom de la DLL dont on veut faire une bibliothèque d'importation, `def` est le nom du fichier `.DEF` généré par **impdef**, et `bibliothèque` est le nom de la bibliothèque d'importation à créer.

La deuxième utilisation de **dlltool** est de générer le fichier `.DEF` pour une DLL en cours de développement. **dlltool** permet en effet de générer le fichier `.DEF` à partir des fichiers objets créés par le

compilateur. Ces fichiers `.DEF` contiennent la déclaration des symboles exportés dans le fichier objet. Cette fonctionnalité de **dlltool** est très pratique pour exporter des symboles C++, dont le nom est décoré par le compilateur.

Pour que cette opération fonctionne correctement, il est nécessaire que les symboles qui doivent être exportés aient été déclarés avec l'attribut `dllexport` dans le fichier source. Ceci se fait simplement à l'aide du mot-clé `__attribute__` :

```
__attribute((dllexport))
```

> **Note :** Les doubles parenthèses sont nécessaires.
>
> La macro `__declspec` est également utilisable, elle est définie pour la portabilité avec le mot-clef du même nom dans Visual C++.

Une fois le fichier compilé, il suffit d'appeler **dlltool** avec la syntaxe suivante :

```
dlltool objet --output-def def
```

où `objet` représente le fichier objet contenant la définition des symboles exportés, et `def` le nom du fichier `.DEF` à générer.

> **Note :** Notez que d'une manière générale, si les fonctions à exporter sont définies dans plusieurs fichiers objets, il faudra générer plusieurs fichiers `.DEF` et les fusionner.

La troisième et plus importante utilisation de **dlltool** est la création de la table des symboles exportés, que le linker utilise pour créer la DLL. Cette opération nécessite le fichier `.DEF` contenant la liste des symboles exportés par la DLL, et éventuellement le fichier d'information généré par le linker pour le relogement de la DLL (il est impératif de faire des DLL relogeables, faute de quoi on s'expose à de graves problèmes lors de l'initialisation des programmes qui les utilisent).

La syntaxe à utiliser est la suivante :

```
dlltool --dllname dll --base-file base --output-exp exp --input-def def
```

où `dll` est le nom de la DLL (utilisé dans le fichier d'exportation), `base` est le nom du fichier d'information de relogement généré par le linker, `exp` est le nom du fichier d'exportation à créer et `def` est le nom du fichier d'importation `.DEF`.

Le fichier d'exportation ainsi créé doit alors être passé directement, comme un fichier objet normal, dans la ligne de commande du linker pour être lié avec le code objet de la DLL.

> **Note :** Normalement, les fonctions exportées par les DLL suivent la convention d'appel `STDCALL` de Windows, qui est en fait la convention d'appel standard du Pascal. Cependant, il n'est pas nécessaire d'utiliser ces conventions, sauf si l'on veut utiliser la DLL avec un autre compilateur.
>
> Par ailleurs, la convention de nommage classique des fonctions exportées est le nom de la fonction, sans '_' en préfixe, et avec le suffixe « `@n` », où `n` est le nombre d'octets passés sur la pile. Certaines DLL emploient une convention de nommage différente, où seul le nom de la fonction est utilisé. Les fichiers `.DEF` générés par l'utilitaire **impdef** pour ces DLL contiennent donc toujours le nom exact des fonctions exportées dans la DLL passée en paramètre, c'est à dire avec « `@n` » ou sans selon le type de DLL.

Le problème est que le compilateur utilise toujours le nom décoré de « @n » d'une fonction dès lors qu'elle est déclarée avec STDCALL ou PASCAL. La bibliothèque d'importation générée par **dlltool** à partir de ce fichier .DEF ne permet donc pas de résoudre les symboles lors de l'édition de liens des programmes avec cette bibliothèque.

Ce problème peut être résolu en rajoutant les « @n » manquant aux noms de fonctions du fichier .DEF généré par **impdef**. La bibliothèque d'importation créée par **dlltool** résoudra alors tous les symboles de la DLL lors de l'édition de liens du programme qui l'utilise. Cependant, cette technique provoque une erreur à l'exécution, parce que cette fois, les symboles appelés par les fonctions d'importation de la bibliothèque statique d'importation utiliseront les symboles « @n » pour appeler les fonctions de la DLL, or ces fonctions n'ont pas ces symboles dans leurs noms... C'est pour résoudre ce deuxième problème que **dlltool** dispose de l'option -k, qui permet de supprimer les symboles « @n » du nom des fonctions dans la bibliothèque statique d'importation, mais de les conserver pour les noms des fonctions d'importations en interne. Ainsi, la bibliothèque d'importation est liée correctement au programme qui utilise la DLL, et les symboles de la DLL sont correctement résolus à l'exécution.

Un autre problème peut se poser lorsque l'on cherche à réaliser une DLL dont les fonctions exportées n'ont pas les symboles « @n », car les noms des fonctions générées par le compilateur ont toujours ces symboles. La solution consiste cette fois à exporter les fonctions des DLL sous un autre nom. Pour cela, il faut spécifier les noms sous lesquels les fonctions seront exportées dans le fichier .DEF, et indiquer leurs noms réels à l'aide de la syntaxe suivante :

```
EXPORTS
    export=nom@n
```

où export est le nom sous lequel la fonction de la DLL doit être exportée, et nom@n est le nom réel de la fonction dans le code objet de la DLL (c'est à dire le nom utilisé par le compilateur). Ceci peut se faire simplement en éditant le fichier .DEF généré par **dlltool** à partir du fichier objet contenant les fonctions à exporter, pour rajouter la définition des alias pour l'exportation. L'option -A de **dlltool** permet de faire en sorte que **dlltool** génère automatiquement ces alias lors de la création des fichiers de définition à partir des fichiers objets. La DLL ainsi générée exportera chacune des fonctions sous les deux noms, avec et sans les symboles « @n ». L'option -k fonctionne toujours lors de la création de la bibliothèque d'importation, elle permet de n'utiliser que les noms sans ces symboles.

Chapitre 14. Le débogueur

gdb est un débogueur complet permettant de travailler au niveau langage machine et au niveau source, pour les langages C et C++. Pour qu'un programme puisse être débogué avec **gdb**, il faut que son fichier exécutable contienne des informations symboliques de déboguage. Ces informations peuvent être générées si l'option `-g` est passée au compilateur et si ces informations n'ont pas été retirées de l'exécutable généré par l'éditeur de lien à l'aide du programme **strip**.

14.1. Chargement d'un programme

Le chargement d'un programme au lancement de **gdb** se fait simplement en passant le nom du programme en ligne de commande :

```
gdb programme
```

Lorsque **gdb** se lance, le programme est chargé en mémoire mais n'est pas lancé.

Le chargement d'un fichier core peut être réalisé en précisant le nom du fichier core à la suite du nom du programme. Les informations symboliques de déboguage seront lues à partir du fichier exécutable :

```
gdb programme core
```

SPÉCIFIQUE DOS : Les fichiers core n'existent pas sous DOS.

SPÉCIFIQUE WIN32 : Il est possible de générer des fichiers core sous Windows NT, 2000 ou XP à l'aide de l'outil Dr. Watson. Cependant, **gdb** ne sait pas lire ces fichiers. Pour cela, vous devrez utiliser le débogueur WinDBG, fourni en standard avec le Platform SDK de Windows.

Pour s'attacher à un programme, il est nécessaire de connaître son PID. La ligne de commande suivante peut alors être utilisée :

```
gdb programme PID
```

Il est également possible de charger un programme une fois **gdb** démarré. Pour cela, on utilisera la commande suivante :

```
file exécutable
```

De même, l'attachement à un processus depuis **gdb** se fait à l'aide de la commande suivante :

```
attach PID
```

Il est nécessaire d'utiliser la commande **file** pour charger le fichier exécutable avant de s'attacher à un processus, faute de quoi les informations symboliques de déboguage ne seront pas disponibles. Enfin, pour se détacher d'un programme, il suffit de taper **detach**.

La lecture d'un fichier core depuis **gdb** peut être réalisée avec la commande suivante :

```
core-file <core>
```

Comme pour l'attachement, cette commande suppose que le fichier exécutable ait été chargé au préalable avec la commande **file**.

> **Note :** Si l'exécutable et les informations symboliques de déboguage ne sont pas dans le même fichier, il est possible d'utiliser les commandes suivantes pour spécifier ces fichiers indépendamment l'un de l'autre :
>
> ```
> exec_file exécutable
> symbol_file symboles
> ```

14.2. Commandes fondamentales

Une fois **gdb** démarré, il est possible de lancer l'exécution du programme à l'aide de la commande **run paramètres**, où `paramètres` représente les options en ligne de commande du programme. La ligne de commande peut être lue à tout moment à l'aide de la commande **show args**. Pour terminer un programme en cours d'exécution, il faut utiliser la commande **kill**. Pour quitter **gdb**, il suffit de taper **quit**.

gdb dispose d'un grand nombre de commandes, que l'on saisie interactivement en fonction de l'état du programme en cours de déboguage. Cependant, il est également possible de programmer gdb avec un fichier de commande, qui lui sera passé en paramètre au lancement à l'aide de la syntaxe suivante :

```
gdb -command fichier
```

14.3. Les points d'arrêt

La commande **run** citée ci-dessus lance l'exécution du programme inconditionnellement. Si le programme ne génère pas d'erreur, il se terminera normalement. Il faut donc utiliser les commandes de point d'arrêts pour stopper l'exécution du programme afin d'examiner son état. **gdb** gère un grand nombre de types de points d'arrêt et permet de contrôler finement leur fonctionnement.

Pour placer un point d'arrêt dans le programme, il faut utiliser la commande **break**. Cette commande permet de poser tout type de point d'arrêt, les formes les plus utiles de cette commande sont données ci-dessous :

Commande	Signification

Commande	Signification
break fonction	Pose un point d'arrêt à l'entrée de la fonction `fonction` du fichier courant.
break fichier:fonction	Pose un point d'arrêt sur la fonction `fonction` du fichier `fichier`.
break <**+offset**> ou **break** <**-offset**>	Pose un point d'arrêt à la position courante plus l'offset précisé en paramètre. Cet offset est exprimé en nombre de lignes.
break <**ligne**>	Pose un point d'arrêt à la ligne `ligne` du fichier courant.
break <**fichier:ligne**>	Pose un point d'arrêt à la ligne `ligne` du fichier `fichier`.
break <***adresse**>	Pose un point d'arrêt à l'adresse spécifiée en langage machine.
break	Pose un point d'arrêt à l'instruction suivante de celle en cours d'exécution. Si l'on se trouve dans un appel de fonction, il est possible de changer le contexte d'appel pour celui de la fonction appelante (voir plus loin pour plus de détails à ce sujet). Dans ce cas, l'instruction suivante sera l'instruction qui suit l'appel de cette fonction. La commande **break** permet donc de placer un point d'arrêt au retour de la fonction en cours d'exécution.

Toutes ces commandes peuvent prendre une condition en paramètre. Cette condition est exprimée à l'aide d'une expression utilisant le langage du programme. Cette condition est évaluée à chaque passage sur le point d'arrêt, et le programme n'est stoppé que si elle est vérifiée. La syntaxe pour poser une condition sur le point d'arrêt est la suivante :

```
break paramètres if condition
```

où `paramètres` représente les paramètres de la commande **break** sans condition, et `condition` est l'expression permettant d'évaluer la condition.

Tous les points d'arrêt se voient attribué un numéro par **gdb**. Il est possible d'utiliser la commande suivante :

```
info breakpoints [n]
```

pour obtenir des informations sur le point d'arrêt `n`. Si le numéro du point d'arrêt n'est pas indiqué, **gdb** affiche ces informations pour tous les points d'arrêt existants.

Il est possible de changer la condition sur un point d'arrêt à l'aide de la commande **condition**. La syntaxe de cette commande est donnée ci-dessous :

```
condition n condition
```

où `n` est le numéro du point d'arrêt, et `condition` la nouvelle condition à utiliser. Si cette nouvelle condition est vide, le point d'arrêt redevient un point d'arrêt non conditionnel.

Pour supprimer un point d'arrêt, il suffit d'utiliser la commande **clear**. Cette commande fonctionne avec la même syntaxe que la commande **break** correspondante. Les principales formes de la commande **clear** sont données ci-dessous :

```
clear <fonction>
clear <fichier:fonction>
clear <ligne>
clear <fichier:ligne>
clear
```

Cette dernière forme de **clear** supprime le point d'arrêt de l'instruction suivante. Comme pour la commande **break**, cette commande agit sur le contexte d'appel en vigueur au moment où elle est utilisée. La commande **delete** est également disponible, elle permet de supprimer des points d'arrêt en spécifiant leur numéro :

```
delete n...
```

Si aucun argument n'est donné à la commande **delete**, elle supprime tous les points d'arrêt existants.

Pour désactiver un point d'arrêt sans le supprimer, il suffit d'utiliser la commande **disable**. La syntaxe de cette commande est donnée ci-dessous :

```
disable n
```

où `n` est le numéro du point d'arrêt à désactiver. Si aucun numéro n'est passé en paramètre, la commande **disable** désactive tous les points d'arrêt existants.

La commande **enable** permet de réactiver ces points d'arrêt. Il est possible d'activer un point d'arrêt de plusieurs manières différentes. Les différentes formes de cette commande sont données ci-dessous :

Commande	Signification
enable liste	Active les points d'arrêt de la liste de points d'arrêt `liste`.
enable once liste	Active les points d'arrêt spécifiés pour un seul passage uniquement. Lorsque ces points d'arrêt seront atteints, ils seront automatiquement désactivés.
enable delete liste	Active les points d'arrêt spécifiés. Lorsque ces points d'arrêt seront atteints, ils seront automatiquement détruits.

Dans toutes ces commandes, `liste` est la liste des numéros des points d'arrêt à activer. Si aucun numéro n'est spécifié, la commande s'appliquera à tous les points d'arrêt existants.

Il est possible de spécifier une condition sur le compteur de passage d'un point d'arrêt. À chaque fois que ce point d'arrêt sera rencontré, le compteur de passage sera incrémenté. Le point d'arrêt ne devient valide que lorsque la limite sur le nombre de passage est atteinte. Cette limite peut être fixée à l'aide de la commande suivantes :

```
ignore n compte
```

où `n` est le numéro du point d'arrêt et `compte` est le nombre de passages qui doivent se faire avant que le point d'arrêt ne soit activé.

gdb permet de définir une série de commande à effectuer pour un point d'arrêt lorsque celui-ci est atteint (et si la condition qu'il contient est vérifiée). Cette série de commande peut être spécifiée à l'aide de la syntaxe suivante :

```
commands n
c1
c2
 ⋮
cn
end
```

où `n` représente le numéro du point d'arrêt, et `c1`, `c2`, etc... les commandes à exécuter. Pour supprimer une série de commande d'un point d'arrêt, il suffit de spécifier une série de commande vide.

14.4. Contrôle de l'exécution

Comme on l'a vu ci-dessous, la commande **run** lance l'exécution du programme. On ne peut utiliser la commande **run** qu'une fois pour une instance donnée du programme, si on l'utilise une deuxième fois, le programme est redémarré. Il existe donc d'autres commandes, qui permettent de relancer l'exécution du programme si celle-ci s'est arrêtée.

Un programme peut être arrêté pour trois raisons : soit parce qu'il s'est terminé, soit parce qu'il a atteint un point d'arrêt et que la condition de celui-ci est vérifiée, soit parce qu'il a reçu un signal du système. Dans le premier cas, il faut utiliser à nouveau la commande **run** pour relancer une nouvelle instance de ce programme. Le programme est rechargé exactement au même endroit en mémoire, si bien que les adresses des points d'arrêts sont toujours valides. Dans les deux autres cas, il faut relancer l'exécution. Ceci peut être fait à l'aide d'une des commandes décrites ci-dessous.

La commande **continue** permet de relancer l'exécution du programme jusqu'à ce qu'il soit à nouveau arrêté. La commande **continue** peut prendre en paramètre un nombre entier si le programme vient juste d'être interrompu par un point d'arrêt :

```
continue n
```

Ce nombre `n` indique la valeur du compteur à utiliser pour activer ce point d'arrêt. Ainsi, le programme ne sera à nouveau stoppé par ce point d'arrêt que si l'exécution passe `n` fois dessus.

La commande **step** permet d'exécuter une instruction du programme. La notion d'instruction ici est celle du langage du fichier source où le pointeur d'instruction se trouve. Si l'on veut exécuter une instruction du langage machine, il faut utiliser la commande **stepi**, ou **si** en abrégé. Si l'instruction à exécuter est un appel de fonction, l'exécution se poursuit à l'intérieur de cette fonction.

Les commandes **next** et **nexti** sont similaires à **step** et **stepi**, à ceci près qu'elles n'entrent pas dans les fonctions lorsqu'elles sont utilisées pour exécuter un appel de fonction.

La commande **finish** permet de sortir de la fonction courante, et de s'arrêter avant l'instruction qui suit son appel. Le reste du code de la fonction s'exécute normalement. Si l'on veut sortir immédiatement de la fonction (sans exécuter le code restant jusqu'à la prochaine instruction de retour de fonction), il faut utiliser la commande suivante :

```
return expression
```

où `expression` une expression permettant d'évaluer l'éventuelle valeur de retour de la fonction.

La commande **until** permet de reprendre l'exécution du programme jusqu'à ce qu'il atteigne un point du code placé après l'endroit où se trouve le compteur d'instruction. La syntaxe de cette commande est la même que celle de la commande **break**. L'exécution du programme est automatiquement arrêtée par la commande **until** si la fonction courante se termine.

Si l'on veut placer le compteur d'instruction en un endroit donné, sans relancer le programme, il faut utiliser la commande **jump**. La syntaxe de cette commande est la même que celle de la commande **break**.

Enfin, si le programme a été arrêté par la réception d'un signal, la reprise de l'exécution de ce programme se fait en général par le traitement du signal par le programme. Il est cependant possible d'envoyer un autre signal au programme que celui qu'il a reçu, grâce à la commande **signal** :

```
signal n
```

où `n` est le numéro du signal à envoyer. Si ce numéro est `0`, le programme reprend son exécution comme s'il n'avait pas reçu de signal.

14.5. Les chiens de garde

En plus des points d'arrêt, **gdb** gère la notion de chien de garde. Les chiens de garde permettent d'arrêter l'exécution d'un programme dès qu'une condition est vérifiée. À la différence des points d'arrêts, cette condition est évaluée à chaque instruction du programme et non à l'arrivée sur le point d'arrêt qui définit la condition. Les chiens de garde permettent donc de faire une condition d'arrêt globale sur tout le programme. Il va de soi que les chiens de garde ralentissent l'exécution d'un programme (sauf s'ils sont gérés par un dispositif matériel quelconque), car leur condition doit être évaluée à chaque instruction du programme.

Il existe plusieurs formes de chiens de garde. La forme la plus générale permet d'évaluer une expression à chaque instruction et de stopper l'exécution du programme si cette expression change de valeur. Ceci permet de vérifier la validité d'un invariant dans un programme ou de localiser les instructions qui modifient certaines variables. La syntaxe pour ce type de points d'arrêts est donnée ci-dessous :

```
watch expression
```

La deuxième forme de chien de garde permet de stopper l'exécution du programme lorsqu'une donnée est lue (mais pas forcément modifiée). La syntaxe est donnée ci-dessous :

```
rwatch variable
```

Enfin, la troisième et dernière forme permet de stopper l'exécution dès qu'une variable est accédée, que ce soit en lecture ou en écriture. La syntaxe est donnée ci-dessous :

```
awatch variable
```

Ces deux derniers types de chiens de garde ne sont disponibles que sur les plates-formes capables de gérer les points d'arrêt matériel. Ceci implique qu'un nombre limité seulement de tels points d'arrêt peut être créé. En revanche, les chiens de garde créés à l'aide de la commande **watch** sont toujours gérés, car ils ne nécessitent aucun support matériel particulier.

> **SPÉCIFIQUE WIN32 :** Bien que les processeurs ix86 puissent gérer les points d'arrêt et les chiens de garde au niveau matériel, la version Win32 actuelle de gdb n'est pas capable d'utiliser ces fonctionnalités sous Windows. Il n'est donc possible de créer des chiens de garde qu'avec la commande **watch**.

Tous les chiens de garde sont traités comme des points d'arrêt par **gdb**. Des numéros de points d'arrêt leurs sont donc affectés, et les commandes permettant de supprimer, d'activer et de désactiver les points d'arrêt sont utilisables sur les chiens de garde. En particulier, il est possible d'obtenir des informations sur les chiens de garde grâce à l'instruction **info watchpoints**.

14.6. Examen et modification des données

Il est possible d'examiner la valeur de toute expression exprimée dans le langage du fichier source. L'expression est évaluée dans le contexte du programme en cours de déboguage, ce qui implique que si elle dispose d'effets de bords, l'état du programme est modifié. La commande pour afficher le résultat d'une expression est **print**, elle s'utilise de la manière suivante :

```
print [/format] expression
```

Le paramètre `/format` est facultatif. Il permet de spécifier le type avec lequel l'expression doit être affichée. Si ce type est indiqué, `format` peut prendre l'une des valeurs suivantes :

Valeur	Signification
`x`	Entier en héxadécimal.
`d`	Entier signé.
`u`	Entier non signé.
`o`	Entier en octal.
`t`	Entier en binaire.
`a`	Adresse.
`c`	Caractère.
`f`	Flottant.

La commande **call** permet d'évaluer l'expression sans afficher le résultat. Elle peut être utilisée pour évaluer les expressions qui ne renvoient pas de valeur, et éviter ainsi l'affichage de `void`. Sa syntaxe est la même que celle de **print**.

gdb peut afficher le résultat d'une expression à chaque arrêt du programme. Ceci permet de tracer l'évolution de la valeur de cette expression au fur et à mesure de l'exécution du programme. Pour tracer une expression, il faut utiliser la commande **display** à la place de la commande **print**. Pour chaque expression tracée, **gdb** affecte un numéro automatique. La commande **info** permet d'afficher la liste des expressions tracées :

```
info display
```

Comme pour les points d'arrêt, il est possible d'activer et de désactiver la trace d'une expression à l'aide des commandes suivantes :

```
enable display n
disable display n
```

où `n` est le numéro de la trace.

Pour supprimer la trace automatique d'une variable, il faut utiliser l'une des commandes suivantes :

```
undisplay n
delete display n
```

Il est possible d'examiner la mémoire du programme avec la commande suivante :

```
x [[/NFT] adresse]
```

où `adresse` est l'adresse de la zone de mémoire à afficher, et `NFT` permet de spécifier respectivement le nombre d'objets à afficher (paramètre `N`), le format avec lequel ces objets doivent être affichés (paramètre `F`), et la taille de ces objets (paramètre `T`). Les formats disponibles sont les mêmes que les formats de la commande **print**. La taille des objets affichés est spécifiée par l'une des valeurs suivantes :

Format	Signification
`b`	Taille d'un octet.
`h`	Taille d'un demi-mot, soit deux octets.
`w`	Taille d'un mot mémoire, soit quatre octets.
`g`	Taille d'un double mot, soit huit octets.

Il est possible d'utiliser deux autres formats avec la commande **x**, qui ne sont pas disponibles avec la commande **print**. Ces formats sont donnés ci-dessous :

Format	Signification
`i`	Instruction langage machine.
`s`	Chaîne de caractère terminée par un caractère nul.

Pour ces deux formats, on ne peut pas spécifier la taille des objets, car elle est variable et déterminée automatiquement par **gdb**.

Le paramètre `/NFT` est facultatif, s'il n'est pas précisé, les mêmes options sont reprises pour la commande **x** suivante. De même, le paramètre `adresse` est facultatif, s'il n'est pas précisé, la commande **x** affiche l'objet suivant avec le même format et en supposant qu'il a la même taille que celle utilisée pour la dernière commande.

Les registres du microprocesseur peuvent être examinés à l'aide de la commande **info registers**. Les registres flottants ne sont pas affichés par cette commande. Si l'on veut consulter leur valeur, il faut utiliser la commande **info all-registers**. Il est également possible de consulter la valeur de quelques registres indépendamment les uns des autres à l'aide de la syntaxe suivante :

```
info registers registres
```

où `registres` est la liste des registres à consulter. Les valeurs des registres sont affichées dans le contexte d'exécution sélectionné. Ceci signifie que les valeurs affichées sont celles qui seraient stockées dans les registres si l'on se trouvait dans la fonction en cours ou dans l'une des fonctions appelantes, selon le contexte d'exécution sélectionné. Voir plus loin pour plus de détails à ce sujet.

gdb définit quatre noms standards pour les registres les plus classiques. Ces noms peuvent être utilisés pour référencer ces registres indépendamment de la plate-forme. Il s'agit du registre `pc` pour le pointeur d'instruction, du registre `sp` pour le pointeur de pile, du registre `fp` pour le pointeur de structure de pile et du registre `ps` pour le registre d'état du processeur.

La modification des données se fait simplement par l'évaluation d'une expression contenant une affectation. Il est également possible d'utiliser la commande **set** :

```
set [variable] variable=valeur
```

où `variable` est la variable à modifier et `valeur` est sa valeur. Par défaut, il n'est pas nécessaire d'utiliser le mot-clé `variable`, mais comme la commande **set** peut être utilisée pour fixer les variables d'environnement, il est préférable de le faire.

La commande **set** doit être utilisée si l'on veut modifier la valeur d'un registre du processeur. Le nom de la variable à utiliser est alors le nom du registre, précédé d'un dollar :

```
set $registre=valeur
```

où `registre` est l'un des noms de registres affiché par la commande **info all-registers**.

14.7. Examen du contexte d'exécution

Le contexte d'exécution est sauvegardé lors de chaque appel de fonction. Classiquement, ceci se fait par stockage dans la pile de certains registres du processeur. En particulier, le registre de sélection de structure de pile est le registre qui sert d'adresse de base pour accéder à toutes les variables locales.

gdb permet de consulter la liste des appelants de la fonction en cours pour déterminer le contexte d'exécution qui a permis d'arriver aux points d'arrêts. Il permet également de choisir un contexte d'exécution donné, c'est à dire de remonter dans la pile des appels, afin de permettre la manipulation des variables locales d'une des fonctions appelantes de la fonction en cours. Les contextes d'exécution sont numérotés par gdb dans l'ordre inverse des appels. Normalement, le contexte d'exécution est toujours celui de la fonction en cours d'exécution, c'est à dire le contexte 0. Le contexte de la fonction appelante est le contexte 1, et ainsi de suite.

Pour changer ce contexte, il faut utiliser la commande suivante :

```
frame n
```

où `n` est le numéro du contexte désiré. La commande **frame** affiche le contexte d'exécution atteint à chaque changement. Pour changer de contexte sans avoir cet affichage, il faut utiliser la commande **select-frame** à la place de la commande **frame**. Il est possible de déterminer le contexte courant à l'aide de la commande **frame** sans arguments. Une description plus détaillée est disponible avec la commande **info frame**.

Les commandes **up** et **down** permettent de remonter et de redescendre dans la pile d'appel d'un certain nombre de contextes d'exécution. Leur syntaxe est donnée ci-dessous :

```
up n
down n
```

où `n` est le nombre de contextes d'exécution dont on veut se déplacer. Comme pour frame, les commandes **up** et **down** affichent le contexte d'exécution atteint à chaque fois. Pour se déplacer sans avoir cet affichage, il faut utiliser les commandes **up-silently** et **down-silently**.

Pour visualiser le contexte d'exécution après un point d'arrêt, il faut utiliser la commande **backtrace** (**bt** en abrégé). Cette commande peut prendre un paramètre :

```
backtrace n
```

où `n` est un entier. Si `n` est positif, les `n` contextes d'exécutions précédents sont affichés. S'il est négatif, ce sont les n premiers contextes d'exécution qui sont affichés.

À tout moment, **gdb** peut afficher les arguments, les variables locales et les gestionnaires d'exceptions C++ disponibles pour le contexte d'exécution sélectionné. Ceci se fait à l'aide des commandes suivantes :

Commande	Signification
info args	Affiche les arguments.
info locals	Affiche les variables locales.
info catch	Affiche les gestionnaires d'exceptions actifs.

> **Note :** Le contexte d'exécution influe sur le résultat des commandes d'affichage des valeurs de registres et sur l'effet de la commande **break**.

14.8. Gestion des threads

gdb prend en charge les programmes multithreadés pour les plates-formes qui disposent de cette fonctionnalité. À chaque fois qu'un processus est stoppé, tous ses threads sont gelés. De même, lorsque le processus est relancé, tous les threads sont relancés. Il n'est donc pas possible de « geler » certains threads et de laisser les autres s'exécuter, car l'exécution des threads est du ressort du scheduler du système sous-jacent.

> **SPÉCIFIQUE LINUX :** Le débogage des threads sous Linux nécessite la version 4.18 ou postérieure de **gdb**. D'autre part, **gdb** récupère des paramètres de linuxthreads en lisant dans la mémoire de celui-ci lorsque le programme à déboguer est chargé. Par conséquent, il est nécessaire (et recommandé de toutes manières) que la libraire C du système soit compilée avec les informations symboliques de débogage.

Lorsque l'on manipule un programme stoppé, le contexte d'exécution courant est celui du thread qui a provoqué l'arrêt du programme. Ceci signifie que les variables locales accessibles sont celles qui se trouvent sur la pile de ce thread, et que les registres du processeur ont pour valeurs celles qui sont associées à ce thread. Ce thread spécial se nomme le « thread courant ».

Bien entendu, il est possible de changer de thread courant, à l'aide de la commande **thread** :

```
thread n
```

où `n` est le numéro que **gdb** a assigné à ce thread. Ce numéro n'est pas forcément l'identifiant utilisé par le système pour ce thread. **gdb** utilise en effet sa propre numérotation. Vous pourrez obtenir la liste des threads du processus à l'aide de la commande suivante :

```
info threads
```

gdb permet également de spécifier les threads pour lequels les points d'arrêts et les chiens de garde sont valides. Par défaut, ceux-ci sont actifs pour tous les threads du processus, mais il est possible de ne définir un point d'arrêt que pour un thread donné. Ceci se fait avec le mot-clé `thread` à la suite de la définition du point d'arrêt (et avant la condition de ce point d'arrêt si elle existe) :

```
break ligne thread n
break ligne thread n condition
```

où `n` est le numéro du thread pour lequel ce point d'arrêt est défini. Les paramètres `ligne` et `condition` spécifient l'emplacement du point d'arrêt et la condition d'activation, comme pour la commande **break** générale.

> **Note :** Les chiens de garde gérés au niveau hardware ne peuvent pas, actuellement, être utilisés dans un programme multithreadé. En particulier, sur les plates-formes qui gèrent les chiens de garde avec un support matériel (dont les plates-formes x86, sauf Windows) utilisent systématiquement des chiens de garde matériels s'il reste des registres de déboguage dans le processeur.

14.9. Gestion des fichiers sources

La commande **list** permet d'afficher les lignes du fichier source correspondantes à l'endroit où l'exécution du programme s'est arrêté. Par défaut, la commande **list** affiche dix lignes, centrées autour de l'instruction en cours. Il est également possible d'afficher dix lignes centrées autour d'un symbole, avec la syntaxe suivante :

```
list symbole
```

La manière de spécifier le symbole `symbole` est la même que pour la commande **break**. Si l'on répète la commande **list**, les dix lignes suivantes sont affichées. La commande **list -** permet d'afficher les dix lignes précédentes.

Le nombre de lignes affichées par la commande **list** peut être fixé à l'aide de la commande suivante :

```
set listsize n
```

Enfin, **gdb** peut afficher une partie du fichier source, à l'aide de la syntaxe suivante :

```
list [début],fin | début,[fin]
```

où `début` et `fin` sont les spécifications des lignes du début et de la fin de la zone à lister.

14.10. Spécification du langage

gdb utilise le langage correspondant à l'extension du fichier source correspondant aux informations symboliques de déboguage dont il dispose. Ceci implique qu'il comprend automatiquement les expressions et la syntaxe de ce langage.

Cependant, si le programme a été compilé par un front-end qui a traduit le code source d'un langage en un autre langage, **gdb** peut se tromper de langage. Les commandes suivantes pourront être utiles :

Commande	Signification
show language	Affiche le langage courant.
set language langage	Fixe le langage manuellement.

Dans la dernière commande, les valeurs acceptées pour `langage` sont `c`, `c++`, `modula-2`, `asm` et `local`. Cette dernière option permet de demander à **gdb** de détecter automatiquement le langage en fonction de l'extension du fichier source. **gdb** ne gère nativement que les langages C, C++ et Modula2.

14.11. Complétion des commandes

Il est inutile de taper les commandes entièrement dans **gdb**. En effet, celui-ci utilise un mécanisme de complétion pour toutes les commandes et pour les symboles du programme, s'il n'y a pas d'ambiguïté. Pour l'utiliser, il suffit de taper le début de la commande et d'appuyer sur la touche de tabulation. La commande sera alors complétée, sauf s'il y a ambiguïté. Dans ce cas, **gdb** émettra un bip, et attendra la suite de la saisie. Pour voir la liste des commandes ambiguës, il suffit d'appuyer à nouveau sur la touche de tabulation. **gdb** recopie automatiquement la saisie après avoir affiché la liste des commandes correspondantes afin de permettre la suite de la saisie. Ce mécanisme de complétion est également valable avec les symboles du programme.

Il est souvent inutile d'utiliser la complétion des commandes, parce que la plupart des commandes peuvent être utilisées même lorsqu'elles sont partiellement saisies. En fait, certaines commandes partiellement saisies peuvent être ambiguës, dans ce cas, c'est la commande la plus couramment utilisée qui est exécutée.

Enfin, la touche Entrée permet en général de répéter la commande précédente, sauf pour certaines commandes pour lesquelles ce comportement serait très gênant.

Chapitre 15. Le profiler

Le compilateur C/C++ GNU peut, lors de la compilation, inclure du code additionnel pour chacune des fonctions afin de tracer le temps que le processeur passe dans celles-ci. Le programme est ainsi instrumenté, ce qui signifie que les résultats obtenus par cette méthode de mesure changent les temps d'exécution réels. Cependant, le code additionnel généré par le compilateur n'influe pas trop pour les fonctions les plus lentes. On peut ainsi déterminer relativement facilement les parties du programme qui doivent être optimisées.

Afin de générer les données de profiling, il faut compiler chacun des fichiers à analyser avec l'option `-pg`. L'édition de lien doit également se faire avec l'option `-pg` si l'on utilise GCC. Cette option sert en fait à préciser un fichier de démarrage différent à l'éditeur de lien, fichier qui permet l'initialisation du code de profiling et l'écriture des résultats lors de la terminaison du programme.

Comme l'écriture des résultats est réalisée dans le code de terminaison normale du programme, il est nécessaire que celui-ci se termine correctement. Il faut donc qu'il appelle la fonction `exit` ou qu'il fasse un `return` dans la fonction principale. S'il se termine en raison d'un signal ou d'une exception, aucune donnée ne sera écrite sur disque.

Une fois le programme compilé, il suffit de l'exécuter. Il va de soi que les résultats dépendront des tâches effectuées par le programme : ils varieront selon les fonctions appelées et selon les paramètres fournis. On notera qu'une même fonction peut s'exécuter plus ou moins rapidement selon les paramètres qu'elle reçoit. La méthode de profiling utilisée ici est globale, on ne pourra donc obtenir que des résultats moyens, sauf si l'on répète plusieurs fois le même cas de test.

Lorsque le programme se termine, il génère un fichier de données nommé `gmon.out`. Si un fichier de même nom existe, il est écrasé. Il n'est pas possible de spécifier un autre nom pour ce fichier. En revanche, il est toujours possible de le renommer et de l'utiliser sous ce nouveau nom par la suite. Ce fichier est écrit dans le répertoire courant du programme au moment où il s'est terminé.

Pour analyser ces résultats, il faut utiliser le programme **gprof**. La ligne de commande de ce programme est donnée ci-dessous :

```
gprof option [exécutable [données]]
```

Parmi les options, on trouvera celles-ci (liste non exhaustive) :

Tableau 15-1. Options du profiler

Option	Signification
`--help`	Affiche l'écran d'aide de gprof.
`--version`	Affiche le numéro de version de gprof.
`-b`	Supprime les commentaires expliquant la signification des données.
`-s`	Consolide les résultats passés en paramètres avec des résultats existants. Le résultat de la consolidation est écrit dans un fichier nommé `gmon.sum`. Si ce fichier existe déjà, les résultats sont consolidés avec ceux déjà présent dans ce fichier. Cette option permet de regrouper des statistiques sur plusieurs cas de test.

À part pour les deux premières options, le nom de l'exécutable doit être fourni en paramètre à **gprof**. Si aucun exécutable n'est donné, `a.out` est utilisé par défaut. Cet exécutable est requis parce que

gprof utilise les informations symboliques de déboguage pour interpréter les données.

Un ou plusieurs fichiers de données peuvent être passés en paramètre. Ces fichiers sont les différents fichiers `gmon.out` générés par l'exécutable (éventuellement renommés). Si aucun fichier n'est passé en paramètre, le fichier `gmon.out` est utilisé par défaut.

SPÉCIFIQUE DOS : Le profiler ne fonctionne pas très bien sous DOS.

Chapitre 16. L'analyseur de couverture

L'analyseur de couverture **gcov** utilise, comme le profiler, une instrumentation du code. Il permet de déterminer le nombre de fois que chaque branche du programme a été exécutée. Il génère un fichier de donnée pour chaque fichier source. Ces fichiers sont nommés comme les fichiers sources, et l'extension « .gcov » leur est ajoutée.

Pour pouvoir utiliser l'analyseur de couverture, il faut avoir compilé le programme avec les options `-fprofile-arcs` et `-ftest-coverage`. Il faut ensuite exécuter le programme. Celui-ci écrira alors des fichiers de données lorsqu'il se terminera. Comme pour le profiler, les résultats dépendront des actions effectuées par le programme. Les résultats sont automatiquement cumulés par le code d'analyse de couverture dans les fichiers de données, ce qui permet de calculer le taux de couverture d'une série de cas de tests. Si l'on veut remettre à zéro le décompte de passage pour chaque branche, il faut supprimer les multiples fichiers de données générés après l'exécution du programme.

Les résultats de ces fichiers de données peuvent être interprétés grâce au programme **gcov**. Ce programme prend en paramètre le nom d'un fichier source et génère un fichier du même nom et portant l'extension .gcov. Ce fichier contient les statistiques de couverture pour ce fichier source.

> **SPÉCIFIQUE DOS :** L'analyseur de couverture n'est pas disponible dans la version actuelle de GCC pour DOS (impossibilité d'avoir plus d'une extension dans un nom de fichier).

Chapitre 17. Programmation de composants COM en C++

Les interfaces de programmation COM ont été définies par Microsoft au niveau binaire. Une interface COM est en réalité un pointeur sur un tableau de pointeurs fonctions, qui prennent toutes en premier paramètre le pointeur sur le premier pointeur. Cette structure est exactement la structure qu'utilise le compilateur Visual C++ pour les classes C++ qui disposent de fonctions virtuelles. Les instances de ces classes contiennent en effet un pointeur sur la table des fonctions virtuelles, fonctions qui elles-mêmes prennent en premier paramètre un pointeur sur l'objet `this` (donc l'adresse du pointeur sur la table de fonctions virtuelles). De ce fait, il est très facile de définir une interface en C++ avec Visual C++ : il suffit de définir une classe abstraite, dont les méthodes virtuelles pures sont les méthodes de l'interface.

Pour des raisons de portabilité entre les différentes plates-formes sur lesquelles il est supposé fonctionner, GCC n'utilise pas cette structure pour les tables de fonctions virtuelles (il utilise un mécanisme un peu plus complexe en fait). Par conséquent, il n'est pas possible de définir des interfaces COM simplement en définissant une classe abstraite.

Cependant, étant donné l'importance des composants COM dans le monde Windows, une extension a été ajoutée à la version Windows de GCC pour permettre la définition des interfaces en C++. Cette extension indique au compilateur qu'il doit utiliser le même mécanisme que Visual C++ pour gérer les fonctions virtuelles d'une classe. Cependant, cette extension n'est activée que classe par classe.

Les classes qui doivent utiliser ce mécanisme doivent être déclarées avec la syntaxe suivante :

```
struct __attribute__((com_interface)) structure
{
    ...
};
```

où `structure` est le nom de l'interface.

L'attribut qui est ainsi spécifié doit être placé entre le mot clef `struct` et le nom de l'interface. Pour des raisons de portabilité, il est préférable d'utiliser les macros du type `DECLARE_INTERFACE`, qui ajoutent automatiquement cet attribut et qui s'utilisent de la même manière qu'avec Visual C++. Enfin, il faut compiler tous les fichiers sources avec l'option `-fvtable-thunks` du compilateur pour autoriser la gestion des tables de fonctions virtuelles.

Ainsi, il est possible de déclarer une interface en C++ avec GCC. Cependant, aucun compilateur de fichier IDL n'est fourni, et GCC n'est pas capable de compiler les fichiers d'en-têtes générés par MIDL (il utilise des extensions spécifiques à Visual C++). Vous pouvez donc utiliser des composants, à condition de redéfinir vous-même les interfaces de manière portable (c'est à dire sans les extensions spécifiques à Visual C++), mais vous ne pouvez pas en créer de nouveau.

Chapitre 18. Conclusion

Les outils de la suite de développement GNU sont puissants, fiables, portables et très souples. Ils disposent d'un grand nombre d'options pour les paramétrer. En revanche, ce sont des produits provenant du monde UNIX, ce qui signifie que leur puissance se paie un peu par l'ésotérisme, et tout est en ligne de commande. Le compilateur est excellent : il comprend très bien la plupart des constructions grammaticales des langages C et C++ et le code généré est performant. Malheureusement, le support partiel de COM ainsi que la grande complexité de la gestion des bibliothèques dynamiques Windows réduit sérieusement l'utilité de GCC sous Windows.

En général, pour simplifier l'utilisation de ces outils, il est recommandé d'utiliser **gcc** et **g++** pour les appeler. Ceci arrange sérieusement les lignes de commande et résout beaucoup de problèmes relatifs aux mauvaises options que l'on peut passer aux outils.

Cependant, il est bon de savoir comment le processus de compilation est réalisé. Si l'on veut avoir un ordre d'idée du travail réalisé par **gcc**, il suffit de l'appeler avec l'option `-v` dans les commandes de compilation et d'édition de liens. Il affichera alors toutes les commandes qu'il utilise pour effectuer le travail demandé. Cette option peut être utilisée pour mettre au point des makefile qui ne fonctionnent pas bien.

Rappelons que tous ces outils sont disponibles sur un grand nombre de plates-formes, ce qui signifie que les programmes créés seront relativement portables. Enfin ils sont absolument gratuits, alors, pourquoi ne pas les utiliser ?

Appendix A. GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters.

A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some

or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.